CC3301 Programación de Software de Sistemas – Control 3 – Semestre Otoño 2019 – Prof.: Luis Mateu

Pregunta 1

El problema del viajante o vendedor viajero responde a la siguiente pregunta: dadas $n+1$ ciudades (enumeradas de 0 a n) y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que inicia en la ciudad 0, visita cada ciudad una vez y al finalizar regresa a la ciudad 0?

La solución óptima se puede obtener en tiempo $O(n!)$. Un algoritmo más eficiente puede hacerlo casi en tiempo $O(2^n)$. En la práctica es demasiado lento buscar la solución óptima si n es grande. La función *viajante* de más abajo es una heurística simple para encontrar una solución aproximada. Recibe como parámetros el número n de ciudades (además de la ciudad 0), la matriz m de distancias entre ciudades ($m[i][j]$ es la distancia entre las ciudades i y j), un arreglo z de tamaño $n+1$ en donde se almacenará la ruta más corta encontrada y un número *nperm*. Esta función genera *nperm* rutas aleatorias, calcula las distancias recorridas y se queda con la ruta más corta, entregando en z cuál es esa ruta y retornando la distancia recorrida. No es la ruta óptima, pero mientras más grande es *nperm*, más se acercará al óptimo.

```
int viajante(int z[], int n, int **m, int nperm) {
  int min= INT_MAX; // la distancia más corta hasta el momento
  for (int i= 1; i<=nperm; i++) {
    int x[n+1];     // almacenará una ruta aleatoria
    x[0]= 0;        // la ruta debe comenzar en ciudad 0
    gen_perm(&x[1], n); // genera en x[1]... x[n] una permutación de 1 2 3 ... n
    int d= dist(x, n, m); // calcula distancia al recorrer x[0] x[1] ... x[n] x[0]
    if (d<min) {    // si distancia es menor a la más corta hasta el momento
      min= d;       // d es la nueva distancia más corta
      for (int j= 0; j<=n; j++)
        z[j]= x[j]; // guarda ruta más corta en parámetro z
    }
  }
  return min;       // la ruta más corta está en z
}
```

Las funciones *gen_perm* y *dist* son dadas. Por ejemplo si n es 4, después de la llamada a *gen_perm(&x[1], n)*, el arreglo x podría ser 0, 4, 1, 3, 2. También podría ser 0, 3, 1, 4, 2, etc. Hay $n!$ permutaciones posibles.

Programe la función *viajante_par* con la misma heurística pero en paralelo en p cores. Recibe los mismos parámetros que *viajante*, más el parámetro p. Use *fork* para generar p nuevos procesos pesados. Cada proceso pesado debe generar *nperm* rutas aleatorias (en total *nperm*p* rutas). Use un pipe para que cada proceso hijo entregue su ruta más corta al padre. El padre se encarga de entregar la mejor de las rutas determinadas por los hijos.

Pregunta 2

Esta pregunta consiste en paralelizar el mismo problema de la pregunta 1 recurriendo a un número variable de computadores de un solo core conectados a Internet, organizados en un esquema cliente/servidor.

El proceso servidor es multi-cliente con threads, corre en *anakena*, acepta conexiones a través del puerto 3000 y se invoca de la siguiente manera:

```
$ ./viajante n nperm
```

Cada cliente (comando *generador*) genera múltiples grupos de 1 millón de rutas aleatorias cada uno, por cada grupo envía la mejor ruta al servidor y luego espera un caracter *'c'* del servivor señalando que continúe la búsqueda. El cliente termina cuando recibe del servidor una 't', que significa que entre todos los clientes se completaron las *nperm* rutas aleatorios (o un poco más).

El siguiente es un ejemplo de uso que muestra el servidor trabajando con 3 clientes. Los clientes pueden llegar en cualquier momento. El despliegue de los comandos se muestra en orden cronológico.

servidor	*cliente 1*	*cliente 2*	*cliente 3*
`$ ./viajante 60 100000000`			
	`$ ./generador`		
`cliente c1 conectado`			
		`$ ./generador`	
`cliente c2 conectado`			
			`$ ./generador`
`cliente c3 conectado`			
`mejor ruta c1= 1000`	`mejor= 1000`		
		`mejor= 900`	
`mejor ruta c2= 900`			
			`mejor= 1100`
`resultado c3 descartado`			
	`mejor= 1200`		
`resultado c1 descartado`			
		`mejor= 850`	
`mejor ruta c2= 850`			
...	...	...	...
`Fin! 100000000 rutas generadas.`			
`Distancia ruta mas corta= 850`			
`Ciudades= 0 5 23 45 10 ...`	$	$	$

Por simplicidad, en el cliente suponga que la matriz de distancias está en la variable global m. No se preocupe por la manera en que llegó ahí. El parámetro n debe recibirlo del servidor.

Programe el servidor (*viajante*) y el cliente (*generador*) de manera que reproduzcan las mismas salidas estándares mostradas en el ejemplo de uso. En el servidor programe toda la función *serv* y solo el inicio de *main* para transferir los parámetros del comando de *argv* a variables globales (no programe desde *j_bind* en adelante). No se preocupe por el término del servidor. Tenga cuidado con los dataraces en el servidor: necesitará un mutex. Para el cliente (*generador*) debe programar toda la función *main* y preocuparse de su término.